Webアプリケーション入門 🚀

PythonとFastAPIでのバックエンド開発入門

Webアプリって何？🤔

インターネットを通じて使うサービスやツールのことだよ！

- **例:**
  - ネットショッピングサイト (Amazon, 楽天など)
  - SNS (X, Instagram, Facebookなど)
  - 動画サイト (YouTube, Netflixなど)
  - ブラウザゲーム

ブラウザ（画面） と、その **「裏側」** で動くシステムでできているんだ。

お店に例えてみよう！ 🏪

Webアプリケーションは、大きく2つの部分に分けられるよ。

- **フロントエンド (お店の表側)**
  - お客さんが見る・触る部分 (見た目、ボタン)
  - 役割：ショーウィンドウ、メニュー、店員さん
  - 技術：HTML, CSS, JavaScript
- **バックエンド (お店の裏側)**
  - お客さんには見えない、サービスを動かす心臓部
  - 役割：キッチン、倉庫、在庫管理
  - **今回はこっちを詳しく見ていくよ！ ✨**

バックエンドってどんな場所？🏭

お店の **「裏側」** や **「キッチン」** をイメージしてね！

お客さん（フロントエンド）からの **注文（リクエスト）** を受けて、
料理（処理） をして、**提供（レスポンス）** する、とっても重要な場所だよ。

バックエンドの仲間たち① Webサーバー 🚪

- **役割: 受付・案内係**
- **仕事**:
  - お客さん（ブラウザ）の最初のアクセス（HTTPリクエスト）を受け取る。
  - 簡単な要求は自分で対応（例: メニュー表示）。
  - 複雑な要求（料理の注文など）はキッチン（APサーバー）に伝える。
  - お店の入口で交通整理もする。
- **代表例**: Apache (アパッチ), Nginx (エンジンエックス)

バックエンドの仲間たち② アプリケーションサーバー 👨‍🍳

- **役割: キッチン・シェフ**
- **仕事**:
  - Webアプリの **メイン** の処理を行う場所！
  - レシピ（プログラムコード）に従って調理（計算、データ加工など）。
  - 冷蔵庫（データベース）と連携して材料（データ）を出し入れする。
  - 出来上がった料理（処理結果）を準備する。
- **主な技術**: Java, Python, Ruby, PHP, Node.js など

バックエンドの仲間たち③ データベース 📦

- **役割: 冷蔵庫・食材倉庫**
- **仕事**:
  - 大切な材料（データ）を保管・管理する場所。
    - ユーザー情報、商品リスト、投稿メッセージなど
  - シェフ（APサーバー）の指示に従って、データを **探したり、保存したり、更新したり** する (CRUD操作)。
- **代表例**: MySQL, PostgreSQL (リレーショナル), MongoDB (NoSQL) など

バックエンドの仲間たち④ API (ちょっと専門的) 📝

- **役割: 伝票・連絡メモ**
- **仕事:**
  - お店の各部門がスムーズに連携するための **窓口**。
  - フロントエンド ⇔ バックエンド
  - バックエンド内の部品同士 (サーバー間)
  - 決まった形式（例: JSON）で情報をやり取りする。
    - 「このデータください！」
    - 「この処理をお願いします！」

バックエンドのお仕事まとめ 💼

ユーザーの見えないところで、こんなお仕事をしているよ！

- 注文（リクエスト）を聞く
- 料理を作る（プログラム実行 / ビジネスロジック）
- 冷蔵庫（データベース）を操作する (データの読み書き)
- 会員証を確認する（認証・認可）
- 出来上がった料理（結果 / レスポンス）をお客さんに渡す

バックエンド開発で使う「道具」🛠️

バックエンドを作るには、こんな道具を使うよ。

- **言葉 (プログラミング言語)**
  - シェフが使う言葉 (コンピューターへの指示)
  - 例: Python, Java, Ruby, PHP, Go, JavaScript (Node.js)
- **便利な調理器具セット (フレームワーク)**
  - 開発を楽にする部品の集まり
  - 例: Django (Python), Spring Boot (Java), Rails (Ruby), Laravel (PHP)
- **冷蔵庫 (データベース)**
  - データの保管場所
  - 例: MySQL, PostgreSQL, MongoDB

まとめ ✨

- Webアプリは「表側 (フロントエンド)」と「裏側 (バックエンド)」が協力して動く！
- バックエンドは見えないけど、サービスの **心臓部** ！ 💪
- データ管理や難しい処理を担当しているよ。

Pythonってどんな言語？

- **とにかく分かりやすい！**
  - 英語に近くて読みやすい文法
  - コードの量が少なくて済む
  - ➡️ **初心者さんが始めやすい理由！**
- **何でもできる！**
  - 🌐 Webサイトの仕組み
  - 📊 データ分析
  - 🧠 AI（人工知能）
  - 🤖 面倒な作業の自動化
  - 🎮 ゲーム開発...

- **便利な「部品」がいっぱい！**
  - 「ライブラリ」と呼ばれる便利な機能集
  - ゼロから作らなくてOK！

Pythonの基本の書き方 ①

まずはコンピューターに言わせよう！ `print()`

括弧の中に表示したいものを書きます。

```
print("Hello, World!") # 文字列は "" か '' で囲む
print(123)             # 数字はそのまま
print("私の名前は" + "AIです") # 文字列をつなげることもできる
```

Pythonの基本の書き方 ②

情報をしまっておく箱：変数

データ（文字や数字）を一時的に覚えておくための「名前付きの箱」

`箱の名前 = 入れたいもの` で使います。

```
message = "こんにちは" # message という箱に「こんにちは」を入れる
age = 20              # age という箱に 20 を入れる

print(message) # 箱の中身を表示 -> 「こんにちは」と表示
print(age)     # 箱の中身を表示 -> 「20」と表示
```

Pythonの基本の書き方 ③

自分や他の人へのメモ：コメント

コードの中に説明を書きたいときに使います。

`#` の後ろに書いた内容は、コンピューターは無視します。

```
# これは全体の説明コメントです
print("計算を始めます") # 画面に表示する行のコメント

result = 10 + 20 # 10と20を足してresultに入れる
# print(result) # この行はコメントなので実行されない
```

Pythonの基本の書き方 ④

情報には種類がある：データ型 (代表例)

変数に入れるデータにはいくつかの「種類」があります。

- **数字:** `10` (整数), `3.14` (小数)
- **文字:** `"こんにちは"`, `'Python'` (文字列)
- **Yes / No:** `True`, `False` (真偽値)
- **複数のまとまり:**
  - `[1, 2, 3]`, `['りんご', 'バナナ']` (リスト - 順番あり)
  - `{'名前': '田中', '年齢': 25}` (辞書 - キーと値のペア)

✨超重要✨ Pythonの「インデント」

ここがPythonの一番の特徴です！

インデントとは？

コードの行頭を**字下げ**すること

なぜインデントが必要？

- Pythonでは、インデントで**コードの「まとまり」（ブロック）**を示します。
- **他の言語** ➡️ `{}` や `end` でまとまりを示す
- **Python** ➡️ **インデントの深さ**でまとまりを示す

```
もし 雨が降ったら:    ← 字下げしない
    傘を持っていく    ← 字下げする（まとまり①）
    長靴を履く        ← 字下げする（まとまり①）
もし 晴れなら:        ← 字下げしない
    帽子をかぶる      ← 字下げする（まとまり②）

外に出かける          ← 字下げしない（まとまり終わり）
```

インデントがないとどうなる？

コンピューターが混乱してエラーに！

「この命令はどのまとまりの一部なの？」と判断できません。

```
# これはダメなコード例！
if True:
print("インデントされていません") # ❌ IndentationError になる！
```

正しく字下げしないと動きません！

インデントのルール

- 同じまとまりの中は、**同じ深さ**で字下げする
- 字下げを**戻す**と、そのまとまりは終わり
- 通常は **半角スペース4つ分** の字下げを使うことが多い
- **タブ**（Tabキー）も使えます

インデントの例①：もし〜なら（`if` 文）

条件によって実行する命令を変えるときに使います。

`if 条件:` の下の**字下げされた部分**が、条件が正しいときに動くまとまりです。

```
score = 70

if score >= 60: # もし score が60以上なら...
    # ここから字下げされた部分がまとまり
    print("合格です！")
    print("よく頑張りました！") # 同じまとまりなので同じ深さ
# 字下げを戻したので、if文のまとまりはここまで
print("テスト結果の確認終了")
```

インデントの例②：繰り返し（`for` 文）

リストの項目ごとに何かしたり、決まった回数繰り返したりします。

`for ～:` の下の**字下げされた部分**が、繰り返されるまとまりです。

```
fruits = ['りんご', 'バナナ', 'みかん']

for fruit in fruits: # fruits の項目を一つずつ fruit に入れて...
    # ここから字下げされた部分がまとまり（繰り返される）
    print("果物：" + fruit)
    print("おいしそう！") # 同じまとまりなので同じ深さ
# 字下げを戻したので、for文のまとまりはここまで
print("果物リストを全部見ました")
```

インデントの例③：自分で命令を作る（`def` 関数）

よく使う一連の処理に名前をつけて、後で呼び出せるようにします。

`def 関数名():` の下の**字下げされた部分**が、その関数が実行されたときに動くまとまりです。

```
# greet という名前の関数を定義します
def greet(name): # name は使うときに渡す情報
    # ここから字下げされた部分が greet 関数のまとまり
    print("こんにちは、" + name + "さん！")
    print("お会いできて嬉しいです！") # 同じまとまりなので同じ深さ

# 関数の定義とは関係ない行（インデントを戻す）
print("プログラム開始")

# 定義した関数を呼び出す（使う）
greet("山田") # greet 関数のまとまりが実行される
greet("佐藤") # もう一度 greet 関数のまとまりが実行される
```

FastAPI ってなんだろう？

Webサイトやアプリの裏側 🤔

- インターネットの向こう側には「サーバー」というコンピューターがある
- スマホやPCは「クライアント」としてサーバーにお願いする
- 例:
  - 商品を探したい！
  - 友達の投稿を見たい！

API は「お願い」と「お返事」の通り道

- クライアントからサーバーへの「お願いの仕方」
- サーバーからクライアントへの「お返事の仕方」
- この**ルールのまとまり**が **API** (Application Programming Interface)

Webフレームワーク は API を作る「道具セット」

- API をイチから作るのは大変...
- よく使う機能をまとめた便利な道具セットが **Webフレームワーク**
- **FastAPI** は、Python言語で API を作るための **モダンで高性能な道具セット** ✨

なぜ FastAPI を使うの？

いくつか良いところがあります！

1. とにかく速い！ 🚀

- FastAPI は処理速度がとても速いのが特徴
- たくさんの人が同時に使っても、サーバーが遅くなりにくい
- 例えるなら、一度にたくさん水が流せる太い水道管！

2. コードを書くのが楽！（開発効率UP）

- Python の「型ヒント」という機能を使うのが得意
- 「ここに数字が入るよ」「ここに文字が入るよ」と教えてあげることで...
  - コードを書くときに間違いに気づきやすい
  - 書くべきコードの量が少なく済むことも

3. API の説明書を自動で作ってくれる！📖

- 作った API がどんなお願いを受け付けて、どんなお返事をするのか…
- その**説明書（ドキュメント）**を自動で綺麗に作ってくれます！
- これがあると、自分や他の人が API を使うときにすごく助かるんです👍

「ブラウザでアクセスしたら『Hello World』と表示される、
一番シンプルな Web API を作る」

始める前に：インストール

FastAPI 本体と、API を動かすサーバーが必要です。

```
pip install fastapi uvicorn
# 開発に便利な機能もまとめて入れるなら
# pip install "fastapi[all]"
```

作成するコード

`main.py` というファイル名で以下のコードを書きます。

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
async def read_root():
    return {"message": "Hello World"}
```

このコードが何をしているか、見ていきましょう。

コード解説: 1行目

```
from fastapi import FastAPI
```

- `from fastapi import FastAPI`
  - `fastapi` という「ライブラリ（便利な機能の集まり）」の中から、
  - `FastAPI` という名前の「APIを作るための道具（設計図）」を持ってくる

コード解説: 3行目

```
app = FastAPI()
```

- `app = FastAPI()`
  - 1行目で持ってきた「APIを作るための道具」を使って、
  - **実際に動かす「私たちの API 本体」**を作り出す
  - 作った API に `app` という名前をつけておく

コード解説: 5行目

```
@app.get("/")
```

- `@app.get("/")`
  - これは「デコレーター」という、すぐ下の関数に**「目印」をつける**もの
  - `app` という API 本体へのリクエストのうち、
  - `GET` という方法で（`.get()`）
  - `/` という場所（`("/")`）
  - に来たら、この目印がついた関数を動かしてね！という設定
  - **例えるなら：**「このテーブル（/）に座って、注文（GET）したお客さんには、このレシピの料理を出す」と指示する

コード解説: 6行目

```
async def read_root():
```

- `async def read_root():`
  - これが、リクエストが来たときに**実行される「関数」**の始まり
  - `def` は関数を作るときの合図
  - `read_root` は関数の名前（自分で決める）
  - `async` は「たくさんのリクエストを同時にさばくためのおまじない」（最初は深く考えすぎなくてOK）
  - 最後の `:` は「ここから関数の本体だよ」という区切り

コード解説: 7行目

```
return {"message": "Hello World"}
```

- `return {"message": "Hello World"}`
  - この関数が**「処理結果」として返す**部分
  - `{"message": "Hello World"}` は Python の「辞書」というデータ形式
    - `message` という名前に対して `Hello World` という値が紐づいている
  - FastAPI はこの辞書を、Web でよく使う `JSON` という形式に**自動で変換して返してくれる**

アプリケーションを実行する

`main.py` があるフォルダで、ターミナル（コマンドプロンプト）を開き、以下のコマンドを実行します。

```
uvicorn main:app --reload
```

- `uvicorn`：作った API を動かす**サーバーのプログラム名**
- `main:app`：**「`main.py` ファイルの中にある `app` という名前の FastAPI アプリを動かして」**という意味
- `--reload`：コードを変えたときに**自動でサーバーを再起動**してくれる便利なオプション

API にアクセスしてみよう！

コマンド実行後、ブラウザを開いて以下の URL にアクセスしてみてください。

`http://127.0.0.1:8000/`

`{"message":"Hello World"}`

と表示されれば成功です！✨

自動生成されるドキュメント

FastAPI のすごいところ！
サーバー実行中に、以下の URL にアクセスすると **API の説明書（ドキュメント）** が自動で生成されています。

- `http://127.0.0.1:8000/docs` (Swagger UI 形式)
- `http://127.0.0.1:8000/redoc` (ReDoc 形式)

どんな API があるか、どんなデータが返ってくるかなどを確認できます。

まとめ

- Webアプリは「表側 (フロントエンド)」と「裏側 (バックエンド)」が協力して動く！
- バックエンドは見えないけど、サービスの **心臓部** ！ 💪
- バックエンドはデータ管理や難しい処理を担当しているよ。
- FastAPI は Python で API を作るための **モダンで高性能な道具セット** ✨
- 簡単なコードで、速くて使いやすい API を作れるよ！